Munge

An authentication service and how we exploit it

Munge Overview

What is Munge?

About Munge

> MUNGE (MUNGE Uid 'N' Gid Emporium) is an authentication service for creating and validating credentials. It is designed to be highly scalable for use in an HPC cluster environment. It allows a process to authenticate the UID and GID of another local or remote process within a group of hosts having common users and groups. These hosts form a security realm that is defined by a shared cryptographic key. Clients within this security realm can create and validate credentials without the use of root privileges, reserved ports, or platform-specific methods.
>
> -- Chris Dunlap

A MUNGE security realm is defined by a shared secret between hosts. Any process can create an authentication cookie which ascertains its uid and gid and is valid for a specific period of time.

The cookie can be verified by a process on any other host in the same MUNGE security realm if the process runs under the same uid and gid or as root.

We achieve secure remote file and shell access by:

- assuring that an operator account has the same uid and gid on all relevant servers,
- remote access is done over Wireguard protected connections,

Remote shell access is done with mrsh, remote file access with diod.

Munge References

Homepage: https://dun.github.io/munge/

mrsh

MUNGE enabled rsh and rlogin

About mrsh

> Mrsh is a set of remote shell programs that use munge authentication rather than reserved ports for security. The code for mrsh is based on the source code for rsh, rshd, rlogin, rlogind, and rcp.
>
> -- CHAOS Development Team

mrsh Installation

Overview

Requirements

1. MUNGE security realm set up
2. Wireguard set up
3. build tools and development libraries
4. source code

Install and Configure

1. Compile
2. Set up the server in place
3. Configure the pam profile
4. Symlink the client

Requirements

```
sudo true
sudo apt-get install -y git build-essential libmunge-dev libpam0g-dev libncurses5-dev
# get the code
cd /opt || read -p continue?
git clone https://github.com/chaos/mrsh.git
# compile
cd /opt/mrsh || read -p continue?
./configure
make
```

Set up the server

We require the mlogin service to be registered and propose port 35805.

```
sudo true
getent services mlogin; [ $? == 2 ] && {
sudo tee -a /etc/services <<EOF
mlogin          35805/tcp
EOF
}
```

Determine the IP of the Wireguard interface. Maybe this helps:

```
sudo true
set `sudo wg show|grep interface`
IF=$2
set `ip addr show dev $2 |grep inet`
WGIP=${2%/*}
```

Setup up the service:

```
sudo true
SVDIR=/etc/sv/mrlogind
LOGDIR=/var/log/mrlogind
# Wireguard interface IP:
[ -n "$WGIP" ] || read -p "WGIP is required!"
# create run files
sudo install -d -m 2775 -g staff $SVDIR
cd $SVDIR || read -p continue?
cat > run <<EOF
#!/bin/sh
exec 2>&1
exec tcpsvd -v $WGIP mlogin /opt/mrsh/mrlogind/in.mrlogind
EOF
chmod +x run
mkdir $SVDIR/log && cd log
cat > run <<EOF
#!/bin/sh
```

```
exec chpst -u log svlogd -t ./main
EOF
chmod +x run
# create log directory
sudo install -d -m2750 -o log -g adm $LOGDIR
cd $SVDIR/log && ln -s $LOGDIR main
# activate service
cd $SVDIR && ln -s `pwd` /service
```

Configure the pam profile

```
sudo true
sudo tee /etc/pam.d/mrlogin <<EOF
#%PAM-1.0
# For root login to succeed here with pam_securetty, "mrsh" must be
# listed in /etc/securetty.

auth       requisite    pam_nologin.so
auth       required     pam_localuser.so
auth       required     pam_shells.so
auth       required     pam_securetty.so

@include common-account

@include common-session
EOF
```

mrsh References

Homepage: https://github.com/chaos/mrsh

diod

MUNGE enabled 9p file server

About diod

> diod is a multi-threaded, user space file server that speaks 9P2000.L protocol.
>
> -- CHAOS Development Team

diod References

Homepage: https://github.com/chaos/diod